A Rigorous Computational Comparison of Alternative Solution Methods for the Generalized Assignment Problem

Mohammad M. Amini • Michael Racer
Department of Management Information Systems and Decision Sciences, The Fogelman College of Business and Economics, The University of Memphis, Memphis, Tennessee 38152
Department of Civil Engineering, The Herff College of Engineering, The University of Memphis, Memphis, Tennessee 38152

Statistical experimental design and analysis is a cornerstone for scientific inquiry that is rarely applied in reporting computational testing. This approach is employed to study the relative performance characteristics of the four leading algorithmic and heuristic alternatives to solve the *Linear Cost Generalized Assignment Problem* (*LCGAP*) against a newly developed heuristic, *Variable-Depth Search Heuristic* (*VDSH*). In assessing the relative effectiveness of the prominent solution methodologies and *VDSH* under the effects of various problem characteristics, we devise a carefully designed experimentation of state-of-the-art implementations; through a rigorous statistical analysis we identify the most efficient method(s) for commonly studied LCGAPs, and determine the effect on solution time and quality of problem class and size.
(*Combinatorial Optimization; Generalized Assignment Problem; Variable-depth Search; Experimental Design and Analysis*)

1. Introduction

One of the fundamental bases for validation of scientific inquiries is a rigorous standard of experimental protocol—including the use of statistical experimental designs—which allows for drawing inferences from the observed data. Unfortunately, this is not a norm in the mathematical programming literature, which often present unclear designs and unreplicated point estimates when reporting empirical testing of software. The absence of a statistically valid, systematic approach can result in the drawing of insupportable conclusions regarding the relative performance of alternative algorithms' and/or heuristics' implementations. The lack of an a priori experimental design is believed to be one of the main sources of such shortcomings (see Crowder, Dembo, and Mulvey (1978), McGeoch (1986), Amini (1989), Greenberg (1990), Jackson et al. (1990), Amini and Barr (1992)).

This paper first presents a new heuristic, *Variable-Depth-Search Heuristic* (*VDSH*), to solve the *Linear Cost Generalized Assignment Problem* (*LCGAP*). Next, it illustrates the application of basic principles of statistical design and analysis of experiments to compare four leading LCGAP alternative solution methodologies with VDSH. The purposes of our carefully designed experiment are to study the relative efficiency of the prominent solution approaches with respect to the problem characteristics and to provide answers to the following questions regarding the alternative solution approaches:

- Is there a best overall method for solving *LCGAP*?
- What are the effects of type and degree of parametric change on the performance of each solution methodology?

0025-1909/94/4007/0868$01.25

- What are the effects of problem set and size on the performance of each method?
- What are the interaction effects on the solution techniques when the above factors are changed singly or in combination?

To fully explore the interaction between LCGAP solution methodologies and salient problem characteristics, we (1) design and implement a new heuristic, *VDSH*; (2) devise a statistical experimental design to evaluate the relative efficiencies of the leading solution methods; (3) create a portable testing system to generate all necessary data points; and (4) implement a rigorous statistical analysis of the performance of the methods under different experimental combinations.

The remainder of the paper is organized as follows: In §2 we present a background on *LCGAP* and a discussion on the alternative solution approaches. Our new heuristic *VDSH* along with an example to illustrate the steps involved in the heuristic are also discussed. The computer implementation and the complexity of the leading methods and *VDSH* are discussed in §3. A systematic test procedure is presented in §4. The experimental design for computational comparison of the *VDSH* with the four leading solution methods is the topic of §5, followed by discussions on the design, implementation, and analysis of results in §6. Finally, the summary and conclusions are presented in §7.

2. Background

In this section, we first state the generalized assignment problem. Next, we review previous methodologies, and introduce in detail the new heuristic, *VDSH*.

2.1. Problem Statement

An important class of network models is the *linear cost assignment problem* (*LCAP*). The assignment problem constitutes a linear programming problem in which a set of *n assignees* (e.g. employees, objects, machines, or sales districts) is to be assigned uniquely to *n* particular *assignments* (e.g. tasks, persons, jobs, or salesmen). Each assignee *i* associates a cost c_{ij} with each assignment *j*. The objective for the assignment problem is to assign one assignee to each assignment in such a way as to minimize the sum of the costs. In the last forty years, a variety of algorithms have been devised and implemented on the traditional as well as novel computers (vector and parallel), which efficiently provide optimal solution to the *LCAP*.

A challenging variation of *LCAP* which has found merit in real-world applications is the *linear cost generalized assignment problem* (*LCGAP*). The *LCGAP* may be posed as follows: assign a set of *m* assignees (e.g. vehicles, employees, etc.), each with a limited capacity (e.g. capacity, speed, etc.), uniquely to a set of *n* particular assignments (e.g. packages, loads, jobs, etc.), each consuming some amount of the assignee's capacity. Each assignee *i* has a capacity b_i and associates a cost c_{ij} with each assignment *j*. Also, each assignment *j* requires an amount r_{ij} of assignee *i*'s capacity to be completed. The *LCGAP* attempts to determine a set of unique assignments without violating any of the assignees' limited capacity.

To present a precise mathematical statement of the *LCGAP* we use the following notation:

Constants:
m: *number of assignees*,
n: *number of assignments*,
b_i: *capacity of assignee i*,
r_{ij}: *assignment j's requirement of assignee i's capacity*,
c_{ij}: *cost of assigning assignee i to assignment j*.
Variables:

$$x_{ij}: \begin{cases} 1, \textit{ if assignee i is assigned to assignment j} \\ 0, \textit{ otherwise}. \end{cases}$$

We may formulate the *LCGAP* as follows:

$$\textit{Minimize} \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} r_{ij} x_{ij} \le b_i, \quad i \in I, \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad j \in J, \tag{3}$$

$$x_{ij} = 0 \textit{ or } 1, \quad i \in I, \quad \textit{and} \quad j \in J. \tag{4}$$

Many applications of *LCGAP* have been reported in the open literature (see Balachandran (1972), Grigoriadis et al. (1974), Ross and Soland (1975), Fisher and Jaikumar (1981), and Racer (1990)). Applications include computer job assignments in computer networks; design of communication networks; some special facility location problems converted to *LCGAP*; vehicle routing, in which the vehicle fleet delivers products stored at a central warehouse to satisfy the orders placed by scattered customers; and software engineering management. A recent study by Gavish and Pirkul (1991) extends the *LCGAP* application to address the *Multi-Resource LCGAP* (*MRLCGAP*), in which assignees have a limited availability of a set of resources. *MRLCGAP* has important application in the trucking industry.

2.2. Methodologies to Compare

Although the *LCGAP* is known to be *NP-hard*, the majority of studies has concentrated on optimization methodologies, applying branch-and-bound method. In this paper, the performance of *VDSH* is compared to that of three optimization methods—Ross and Soland (1975), Fisher, Jaikumar, and van Wassenhove (1986), and Martello and Toth (1987)—and the heuristic of Martello and Toth (1987). We briefly describe each below.

Ross and Soland develop a depth-first branch-and-bound method for the *GAP*. Bounding is achieved in two steps. First, the capacity constraints are relaxed, and assignments are created. Secondly, the bounds are tightened by considering all reassignments within resources with violated capacities. The optimization method of Fisher, Jaikumar, and van Wassenhove is also a depth-first method. Their multiplier adjustment method is a Lagrangian technique relaxing the requirement that each resource be assigned exactly once. Tighter bounds are achieved by intelligently modifying the multipliers.

Martello and Toth's heuristic is a two-pass method. The first pass makes assignments based on minimizing a regret function. This regret may take one of several forms: task cost; task cost/unit weight; task size; task size relative to resource capacity. The second pass of the heuristic identifies any feasible cost-reducing reassignments. The optimization scheme of Martello and Toth utilizes the heuristic as a rough bound. This bound is improved upon by a pair of reduction methods.

2.3. The Variable-Depth-Search Heuristic

In this section, we introduce a new heuristic approach for solving *LCGAP* and to compare its performance with the leading optimization algorithms and heuristics. The present technique relies on the idea of local search (see Papadimitriou and Steiglitz (1982)). Local search methods have been used to provide quality solutions for a variety of problems, including *TSP*, job sequencing problem, etc. (Clarke and Wright (1964), Casco et al. (1988), Hall (1989)). In the case of the *LCGAP*, the capacity constraints were found to hinder the application of the typical search methods. Lin and Kernighan (1973) developed a procedure called *variable-depth search* (*V DS*), to overcome the roadblock in local search methods. Their concept has been successfully applied to both the uniform graph partitioning and traveling salesman problems (Kernighan and Lin (1970), and Lin and Kernighan (1973)). The heuristic presented in this study for solving *LCGAP*, the *variable-depth-search heuristic* (*V DSH*), is motivated by the Lin and Kernighan (1973) traveling salesman heuristic.

The method employed for the GAP is similar to one used by Kernighan and Lin (1970) in solving the uniform graph partitioning problem. The essence of such a method is to begin with some initial, usually feasible, assignment of resources to tasks. From this point, the solution is improved in a step-wise fashion. Heuristics of this sort may be called local search methods. At each step, we seek to find an improvement. If no improvement is possible, then the solution is locally optimal, with respect to the definition of a step.

It is a matter of algorithmic design what is considered to be a step. Possible steps for the GAP may be:

i. reassign a task from one resource to another, or
ii. swap the assignments of two tasks, or
iii. permute the assignments of s tasks, $s \le N$

The implications of these possibilities are important. First, any method that allows only type *i* steps is less flexible than an algorithm that allows type *iii* steps. However, the other prominent factor is the amount of work needed to find an allowable improvement.

Two possible improvement rules are "first improvement" and "best improvement." At each iteration, a first improvement algorithm will scan the set of allowable options until an option is found that will decrease costs. A best improvement rule selects the option that will produce the largest decrease in cost among all options that produce savings. On the average, the best improvement strategy will require more work per iteration than the first improvement type.

All of this is brought back to bear on the three possible steps suggested earlier. The work required to find a type *i* improvement, either first or best, is linear in the number of resources and the number of tasks. Moving from step definition *i* to step definition *iii*, work increases exponentially in s.

The structure of the Variable Depth Search Heuristic is a two-phase algorithm. The first phase develops an initial solution and lower bound. Phase two consists of a doubly-nested iterative refinement process. Within each *major iteration* an action set of potential task moves and swaps is created. Subsequently, the heuristic proceeds through a set of *minor iterations*, to create a *sequence* of actions, in an effort to reduce total costs. The major iteration concludes by identifying the *subsequence* of actions that produces the greatest savings in cost. Thus, a *step* within *VDSH* is an ordered set of moves and swaps, observing capacity constraints, that results in a reduction of costs. If such a sequence is found, the task assignments are revised, and another major iteration is performed. When no such sequence is found, *VDSH* terminates.

2.3.1. Structure of the Variable-Depth Search Heuristic. We define the following variables, functions, and sets:

k = minor iteration counter,

$l(j)$ = resource assignment of task j at start of major iteration,

$\tilde{c}_{i,j} = c_{l(j),j} - c_{i,j}$,

$\hat{c}_{i',j'',i'',j'} = \tilde{c}_{i',j''} + \tilde{c}_{i'',j'}$,

$f(i)$ = sum of sizes of tasks assigned to resource i at start of major iteration,

$e(i,k)$ = sum of sizes of tasks assigned to resource i through minor iteration k,

$p(k)$ = accumulated savings through minor iteration k,

τ_i = set of tasks assigned to resource i at start of major iteration,

$\Theta_{i,k}$ = set of tasks assigned to resource i at minor iteration k,

μ = set of (resource, task) reassignment pairs (i,j) for which task j is not assigned to resource i at start of the major iteration,

σ = set of swap pairs $[(i',j''),(i'',j')]$ for which tasks j' and j'' have different resource assignments—i' and i'' respectively—at start of the major iteration,

$argmax(\cdot)$ = determines the index of the maximum value in a set.

v = the value of adding the best move at the current minor iteration, given all previous actions are also taken

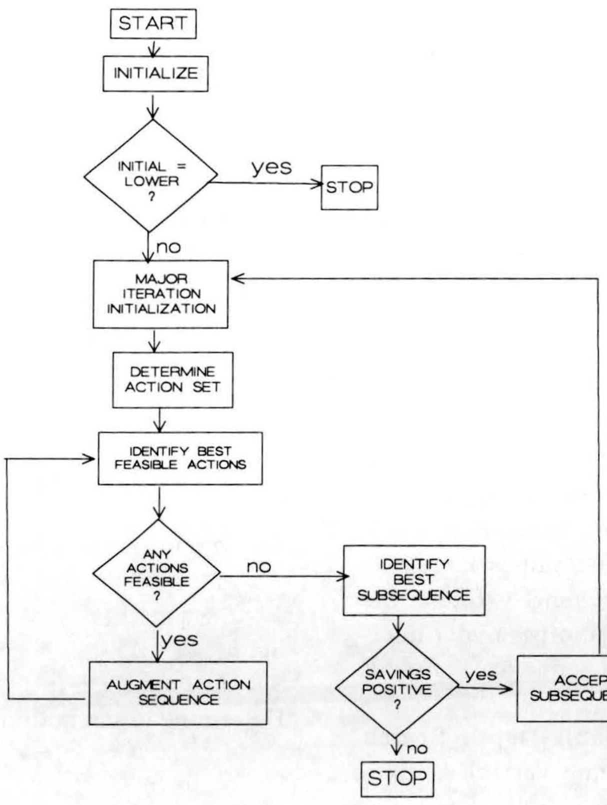
w = the value of adding the best swap at the current minor iteration, given all previous actions are also taken.

We will now describe the GAP Variable-Depth-Search Heuristic (*VDSH*), summarized by the flowchart in Figure 1:

Variable Depth Search Heuristic (VDSH)

```
PHASE I:

Step 0: INITIALIZATION

    Determine an initial partition of the tasks into $\tau_1, \tau_2, \ldots, \tau_{m+1}$;
    $I = I + \{m+1\}$;
    Set $b_{m+1} = \infty$;
    Set $f(i)$, for $\forall\, i \in I$;
    Determine lower bound, using LP relaxation;
    set $g = \sum_{i \in I} \sum_{j \in \tau_i} c_{ij}$

PHASE II:

Step 1: MAJOR ITERATION INITIALIZATION

    Set $\Theta_{i0} = \tau_i$, $e(i,0) = f(i)$, and for each task $j \in \tau_i$, $l(j) = i$ for $\forall\, i \in I$;
    $p(0) = 0$;
    $U = J$;
    Set $k = 0$;

Step 2: a. ACTION SET DETERMINATION

    Determine $\mu = \{(i,j): i \in I, j \in J, j \notin \tau_i\}$;
    Set $\tilde{c}_{ij} = c_{l(j),j} - c_{i,j}$, for each $(i,j) \in \mu$;
    Determine $\sigma = \{[(i',j''), (i'',j')]: i' = l(j') \neq l(j'') = i''\}$;
    Set $\hat{c}_{[(i',j''),(i'',j')]} = \tilde{c}_{i',j''} + \tilde{c}_{i'',j'}$ for each $[(i',j''),(i'',j')] \in \sigma$;
```

Figure 1 ***VDSH* Flowchart**



b. BEST ACTION SET CREATION

```
$v = \max\{ \tilde{c}_{i,j}: (i,j)\in\mu; j\in U; e(i,k) + r_{ij} \le b_i \};$
$\mu^* = \{ (i,j): c_{i,j} = v; (i,j)\in\mu; j\in U; e(i,k) + r_{ij} \le b_i \};$
$w = \max\{ c_{i',j'',i'',j'}: [(i',j''),(i'',j')]\in\sigma; j'\in U; j''\in U;$
  $e(i',k) + r_{i',j''} - r_{i',j'} \le b_{i'}; e(i'',k) - r_{i'',j''} + r_{i'',j'}$
  $\le b_{i''}\}$
$\sigma^* = \{[(i',j''),(i'',j'): w = \hat{c}_{i',j'',i'',j'}; [(i',j''),(i'',j')]\in\sigma;$
  $j'\in U; j''\in U; e(i',k) + r_{i',j''} - r_{i',j'} \le b_{i'}; e(i'',k)$
  $- r_{i'',j''} + r_{i'',j'} \le b_{i''}\};$
$k = k + 1;$
If $\mu^* = \phi$ and $\sigma^* = \phi$, go to Step 3;
```

c. SEQUENCE CREATION

```
$k' = k - 1;$
If $v > w$ then
  Choose $(i^*, j^*) \in \mu;$
  $U = U - \{j^*\};$
  $p(k) = p(k-1) + v;$
  For $\forall\ i\in I$
    If $i = i^*$: $\Theta_{i,k} = \Theta_{i,k'} + \{j^*\};$
                  $e(i,k) = e(i,k') + r_{i^*,j^*};$
    If $i = l(j^*)$: $\Theta_{i,k} = \Theta_{i,k'} - \{j^*\};$
                  $e(i,k) = e(i,k') - r_{i,j^*};$
    Otherwise: $\Theta_{i,k} = \Theta_{i,k'};$
                  $e(i,k) = e(i,k');$
Otherwise $[w \ge v]$
  Choose $[(i'^*, j''^*),(i''^*, j'^*)] \in \sigma^*;$
  $U = U - \{j'^*, j''^*\};$
  $p(k) = p(k-1) + w;$
  For $\forall i\in I$
    If $i = i'^*$: $\Theta_{i,k} = \Theta_{i,k'} - \{j'^*\} + \{j''^*\};$
                  $e(i,k) = e(i,k') + r_{i,j''^*} - r_{i,j'^*};$
    If $i = i''^*$: $\Theta_{i,k} = \Theta_{i,k'} + \{j'^*\} - \{j''^*\};$
                  $e(i,k) = e(i,k') - r_{i,j''^*} + r_{i,j'^*};$
    Otherwise: $\Theta_{i,k} = \Theta_{i,k'};$
                  $e(i,k) = e(i,k');$
Go to *Step 2b*.
```

```
Step 3: SOLUTION REFINEMENT
  $p^* = \max\{p(0), p(1), \ldots, p(k)\}$;
  $s^* = \operatorname{argmax}\{p(0), p(1), \ldots, p(k)\}$;
  If $s^* > 0$
      $g = g - p^*$;
      $\tau_i = \Theta_{i,s^*}$, and $f(i) = e(i, s^*)$ for $\forall i \in I$;
      Go to Step 1;
  Otherwise
      $g$ is cost of current solution;
      $\{\tau_i, \tau_2, \ldots \tau_m\}$ is final partition;
      If $\tau_{m+1} \neq \phi$, then final solution is infeasible;
      Terminate!
```

Initialization. The initial assignment of tasks to resources proceeds as follows. A random permutation of the tasks, $\{j_{\rho(1)}, j_{\rho(2)}, j_{\rho(3)}, \ldots, j_{\rho(n)}\}$, is generated. A dummy resource, $i = m + 1$, is introduced, where the cost of such an assignment is infinite. Task $j_{\rho(1)}$ is assigned to the first assignee with available capacity. If none are available, the task is assigned to the "dummy" assignee, $m + 1$. Subsequent tasks are assigned in a cyclic fashion. That is, suppose $j_{\rho(r)}$ has been initialized to resource h. Then, the assignment of $j_{\rho(r+1)}$ would first be sought with resource $h + 1$, or resource 1 if $h = m$, or $h = m + 1$. We will discuss the rationale for this initialization in detail, after describing the heuristic. Also during the initialization, a lower bound is calculated (by linear programming relaxation), and the cost of the initial solution is determined.

Major Iteration Initialization. The major iteration proceeds by setting the temporary loads, Θ, for the resources, and initializing the task assignment vector, l. The set U consists of all unlabeled tasks—those tasks which have not been used during the iteration. Initially, U includes the total task set, J.

Action Set Determination. In performing a search for an improved solution, the heuristic allows two types of actions—movement of a task to a new resource, and swapping the resource assignments of two tasks. Calculations are made for each task/assignee pair, (j, i), $i \neq l(j)$. $\tilde{c}_{i,j}$ indicates the savings accrued by *moving* task j from $l(j)$ to i. Similarly, for each pair of potential reassignments, $[(i', j''), (i'', j')]$, we calculate $\hat{c}$. $\hat{c}_{[(i',j''),(i'',j')]}$ reflects the accrued savings achieved as a result of *swapping* the assignments of tasks j' and j''. At this point, the algorithm proceeds to the first minor iteration within the major iteration.

Within the major iteration, a sequence of moves and swaps is generated. A task is unlabeled until it has been involved in some action—either a move or a swap—within the major iteration. A move of j to i is feasible at minor iteration k if the task j is unlabeled, and the reassignment of j to i does not exceed the resource limit of i. Likewise, a swap of j' and j'' is feasible if j' and j'' are unlabeled, and the swap does not force either resource limit to be exceeded.

Best Action Set Creation. The *best move* at minor iteration k is a pair (i, j) that maximizes $\tilde{c}_{i,j}$ over all feasible moves. The set μ^* contains all such (i, j) pairs. The *best swap* is $[(i', j''), (i'', j')]$ that maximizes $\hat{c}_{[(i',j''),(i'',j')]}$ over all feasible swaps. The set σ^* contains all such $[(i', j''), (i'', j')]$. Note that the best move and/or swap at iteration k may actually increase costs, i.e. $\tilde{c}_{i,j} < 0$ for $(i, j) \in \mu^*$, and $\hat{c}_{[(i',j''),(i'',j')]} < 0$ for $[(i', j''), (i'', j')] \in \sigma^*$. It is this relaxation of strict descent that enhances the performance of *VDSH*. An action that increases costs may simultaneously make available enough space so that some other significant cost saving action can follow it in the sequence. The example to be discussed later highlights this aspect. If it is not possible to extend the sequence, i.e. $\mu^* = \phi$ and $\sigma^* = \phi$, then the major iteration is concluded by identifying the best subsequence, and refining the solution.

Sequence Creation. If $v > w$, then costs can be best reduced by making a move. A pair $(i^*, j^*) \in \mu^*$ is accepted as the next action in the sequence, v is added to $p(k - 1)$ to determine $p(k)$, resource usages are updated, and j^* is labeled. If on the other hand, the swap is more profitable, then a swap $[(i'^*, j''^*), (i''^*, j'^*)] \in \sigma^*$ is accepted, w is added to $p(k - 1)$, resource usages are updated, and j'^*, j''^* are labeled. The next minor iteration is begun.

Solution Refinement. The major iteration ends when no further feasible moves or swaps exist. The variable $p(k)$ indicates the reduction in costs achieved by performing *all* accepted moves and swaps up to iteration k. The variable p^* identifies the maximum value that $p(\cdot)$ takes on within the current major iteration, and s^* indicates the value of k for which p^* is maximized. If $s^* = 0$, i.e. no cost-reducing sequence was identified, the algorithm halts. If s^* is not zero, then all moves and swaps in the subsequence are made permanent. The next major iteration will begin with each resource i

carrying all tasks in Θ_{i,s^*}. The remaining moves and swaps are ignored.

We return now to a discussion of the initialization. Developing a randomized initialization allows the algorithm to address two issues—feasibility and optimality. The task of identifying a feasible solution to LCGAP is itself an NP-Hard problem. In some instances, a single execution of *VDSH* will not produce a feasible set of assignments—there will be some tasks that are assigned to resource $m + 1$. By re-solving the problem with a different initialization, we can develop some confidence of whether the original problem is feasible. Empirical evidence indicates that a feasible solution to a feasible problem will be generated within a very small number of iterations. In most cases, the first attempt will in fact generate a feasible solution.

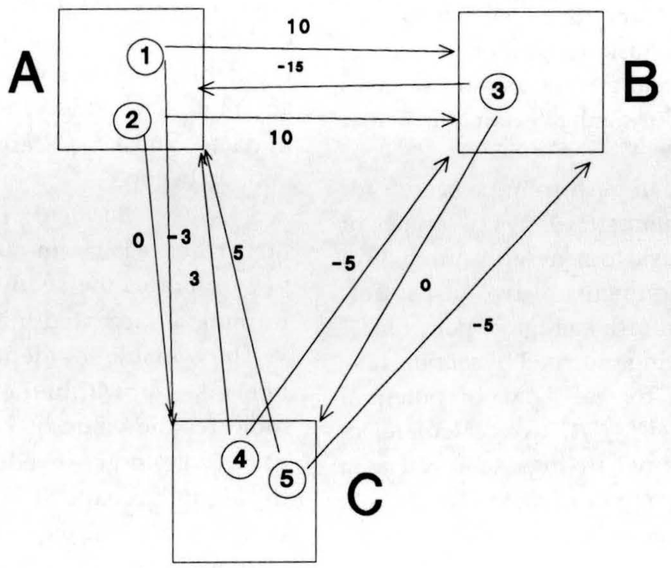
In terms of optimality, the initialization is also a very powerful tool. Because *VDSH* is a heuristic, optimality is not guaranteed. By generating q distinct initializations and solving the LCGAP q times, a deeper understanding of the algorithm's performance may be realized. This concept of multiple initializations was employed by Lin (1973), in motivating the use of the 3-change heuristic for the Traveling Salesman Problem. Being an integer problem, a duality gap will likely exist between the LP lower bound, and the optimum LCGAP solution. By examining the results of the q trials, some confidence can be developed in terms of the size of the duality gap. At the same time, of course, confidence in the solution generated by *VDSH* is also gained.

For any instance of the LCGAP, there are three possible outcomes for a single execution of *VDSH*. First, *VDSH* may be able to identify a solution improvement. Secondly, as discussed earlier, the heuristic may be unable to produce a feasible solution. This is a rare occurrence, overcome by repeating from a new initialization. The third possibility is that there may be a degeneracy, in that a series of tasks can be reassigned with

(a) Table 1 Data Initialization

	Task (j)					
Assignee (i)	1	2	3	4	5	Capacity (b_i)
A	20	20	25	7	5	3
B	10	10	10	10	10	3
C	28	20	10	15	15	3
Requirement (r_j)	2	1	2	1	1	

(b)



no accompanying reduction in cost. In this case, the possibility of non-termination exists. *VDSH* has been written to allow for degenerate improvements, permitting the changes. In this way, *VDSH* allows for the possibility that the change may lead to an actual improvement in the next major iteration. If no improvement is made after a preset number of iterations—currently two—the algorithm terminates. Degeneracy has not proven to be an issue in most instances.

2.3.2. An Example Problem. For simplicity, assume $r_{ij} = r_j$ for this example. Such a situation may, for instance, occur when the tasks are loads to be delivered, and the resources are vehicles. The data is as follows. There are three resources, A–C, each with a capacity of three. The resource requirement vector r is $(2, 1, 2, 1, 1)$. The cost matrix is shown in Table 1(a).

The initial assignments are shown in the Table 1(b). For each task j, and resource i, the value on arc (i, j) indicates $\tilde{c}_{i,j}$. Table 2(a) identifies the action set. The top left table identifies the action set. All potential moves and swaps for the major iteration are shown in the left column. A '/' indicates that such an action is invalid; the capacity of at least one resource would be violated. A '-' indicates that at least one of the tasks involved in the action has already been reassigned in the major iteration. The complete action sequence is depicted in Table 2(b), and the algorithm progress is summarized in Table 2(c).

Had strict monotonicity been forced on the algorithm, the last two operations would not have been performed. The swap of 3 and 4 at the second iteration provided resource B with enough capacity to serve task 1. Also, had the move at the third iteration been one such that savings was negative, then the algorithm would still have retained enough information to know that the only profitable operation was at iteration 1 $(p^* = 5)$. The first major iteration is completed. An improvement has been found. The algorithm proceeds to the second major iteration, with the assignment of task 5 to A, tasks 1 and 4 to B, and tasks 2 and 3 to C.

Table 3 indicates the activity of major iteration 2. After the fifth minor iteration, no actions are possible. The moves of 4 to A, and 2 to B are finalized. The new assignments are tasks 4 and 5 to A, tasks 1 and 2 to B, and task 3 to C.

The algorithm continues with iteration 3, summarized in Table 4. A sequence of four moves is identified. The subsequence, moving task 2 to resource B, is accepted, producing a savings of 10. This is the final cost saving iteration. Iteration four will terminate *VDSH* with the global optimum solution.

3. Computer Implementation of Methods

3.1. Previous Methods

Since computer codes were unavailable for two of the methodologies under comparison—Fisher et al and Ross and Soland—we focus our discussion on the codes of Martello and Toth. In Martello and Toth's optimization method, *MTG*, a total of 67 arrays are defined. Of these, 19 are of length $O(m)$, 29 are of length $O(n)$, and 7 are of order $O(mn)$. In addition, a number of arrays are defined for use in the branch-and-bound structure. Martello and Toth define the value *jnlev* to characterize the number of levels in the tree. Given this, there are 6 additional arrays of $O(jnlev)$, five of length $O(jnlev{*}m)$, and one of length $O(jnlev{*}n)$. Total memory usage is $O(mn + jnlev{*}n + jnlev{*}m)$.

Martello and Toth's heuristic, *HGAP*, is a much simpler procedure, and hence requires much less memory. Total memory for the two-pass method uses sixteen arrays requiring a total of $6m + 7n + 3mn$ in storage.

3.2. VDSH Implementation

In considering implementation, it is apparent that the most computationally intensive operations are the creation of the action set within each major iteration, and the best action set creation within each minor iteration. The action set is actually constructed as two arrays—one for moves, and another for swaps. For each task, all potential reassignments are evaluated and stored. This produces an array of length $O(mn)$. Subsequently, a similarly styled swap array is created, evaluating all potential swaps. The result is an array of length $O(n^2)$ elements.

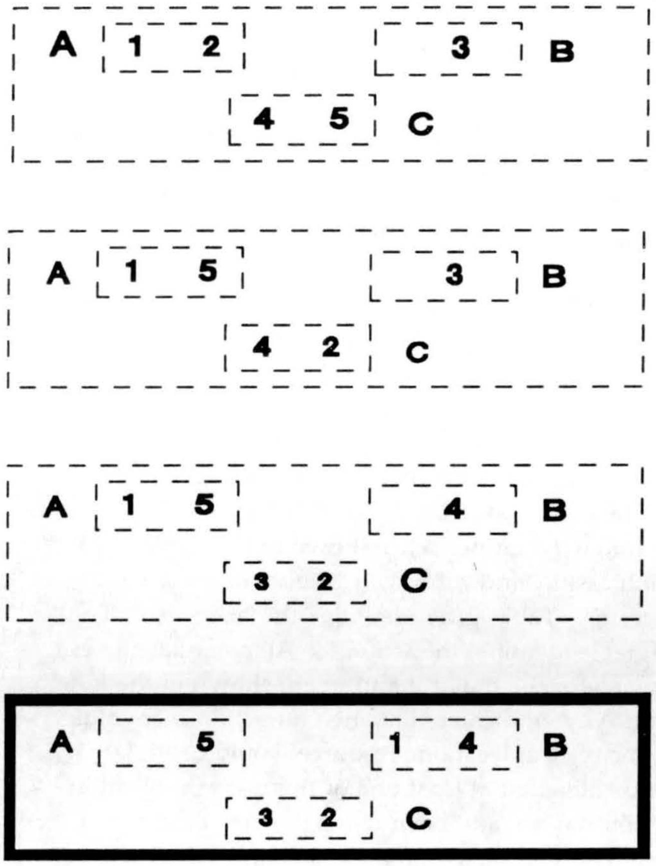
Identifying the best move and swap at each minor iteration requires a scan of each of the above arrays. Two alternatives were considered. The two arrays could be sorted, nondecreasing, requiring time $O(mn \ln mn)$,

Table 2 Major Iteration #1

(a) Action Set

	K		
Moves	1	2	3
1B	/	/	10
1C	/	/	/
2B	10	—	—
2C	0	—	—
3A	/	/	—
3C	/	/	—
4A	/	/	—
4B	−5	−5	—
5A	/	—	—
5B	−5	—	—
Swaps			
13	−5	−5	—
14	−5	−5	—
15	−3	—	—
23	/	—	—
24	3	—	—
25	5	—	—
34	−5	−5	—
35	−5	—	—

(b)



(c) Algorithm Progress

Minor Iteration (k)	Best Move	Best Swap	Chosen Action	Save	p_k	p^*
1	2C	25	Swap	5	5	5
2	4B	34	Swap	−5	0	5
3	1B	—	Move	10	10	10

and $O(n^2 \ln n)$, respectively. Alternatively, the data could remain unsorted. This was the chosen approach. Both arrays are maintained as linked lists. When a task is involved in a move or swap, link pointers are modified so that no further scan is made of actions involving that task. Our motivation for choosing the second requires consideration of the tradeoff between sorting and scanning. Although sorting would place all large savings actions at the top, a large portion of the list might still have to be scanned, searching for a capacity-feasible action. By employing the linked list, the number of elements scanned decreases significantly with each minor iteration.

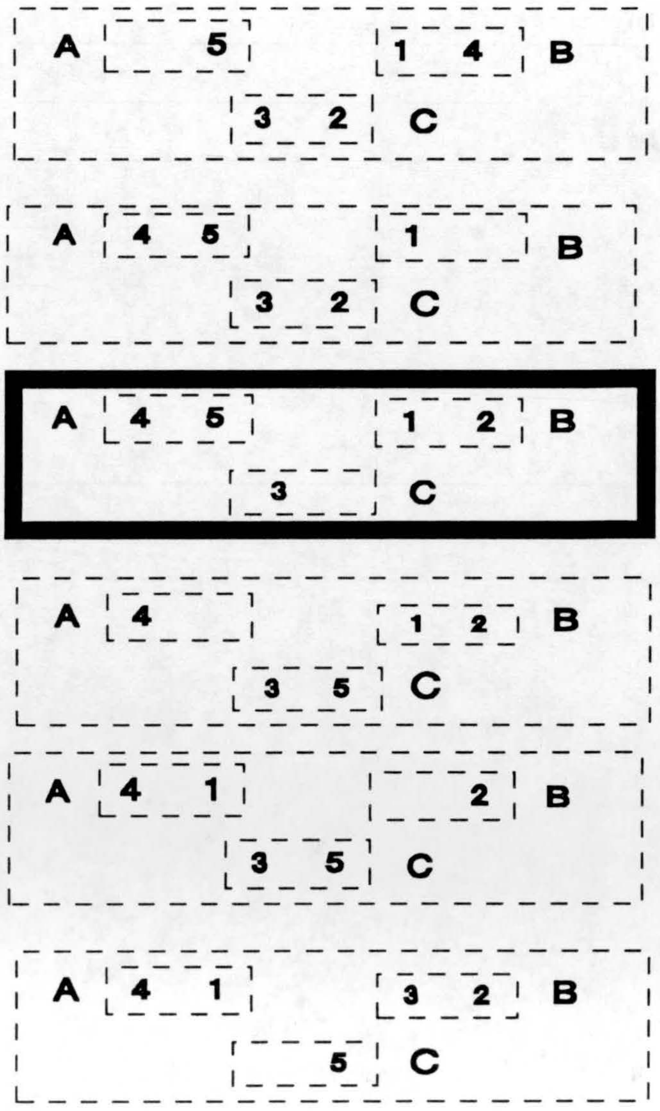
The *VDSH* heuristic requires two input $m \times n$ arrays,

Table 3 Major Iteration #2

(a) Action Set

	K				
Moves	1	2	3	4	5
1A	−10	/	/	−10	——
1C	/	/	/	/	——
2A	0	0	——	——	——
2B	/	10	——	——	——
3A	−15	/	/	−15	/
3B	/	/	/	/	0
4A	3	——	——	——	——
4C	/	——	——	——	——
5B	−5	−5	/	——	——
5C	−10	/	−10	——	——
Swaps					
12	/	/	——	——	——
13	−18	−18	−18	−18	——
15	−15	−15	−15	——	——
24	2	——	——	——	——
25	−10	−10	——	——	——
34	/	——	——	——	——
35	−25	−25	−25	——	——
45	−2	——	——	——	——

(b)



(c) Algorithm Progress

Minor Iteration (k)	Best Move	Best Swap	Chosen Action	Save	p_k	p^*
1	4A	24	Move	2	2	2
2	2B	25	Move	−10	12	12
3	5C	15	Move	−10	2	12
4	1A	13	Move	−10	−8	12
5	3B	——	Move	0	−8	12

storing demand and cost information. Three m-vectors are utilized, indicating resource utilization—current and intermediate—and capacity. Two n-vectors indicate label status and resource assignment for each task. The move set is maintained in an $mn \times 3$ matrix, accompanied by an mn-vector, for the linked list. Similarly, the swap set is contained in an $n^2 \times 3$ matrix, with corresponding n^2 linked list. The selected action sequence is stored in an $n \times 3$ matrix.

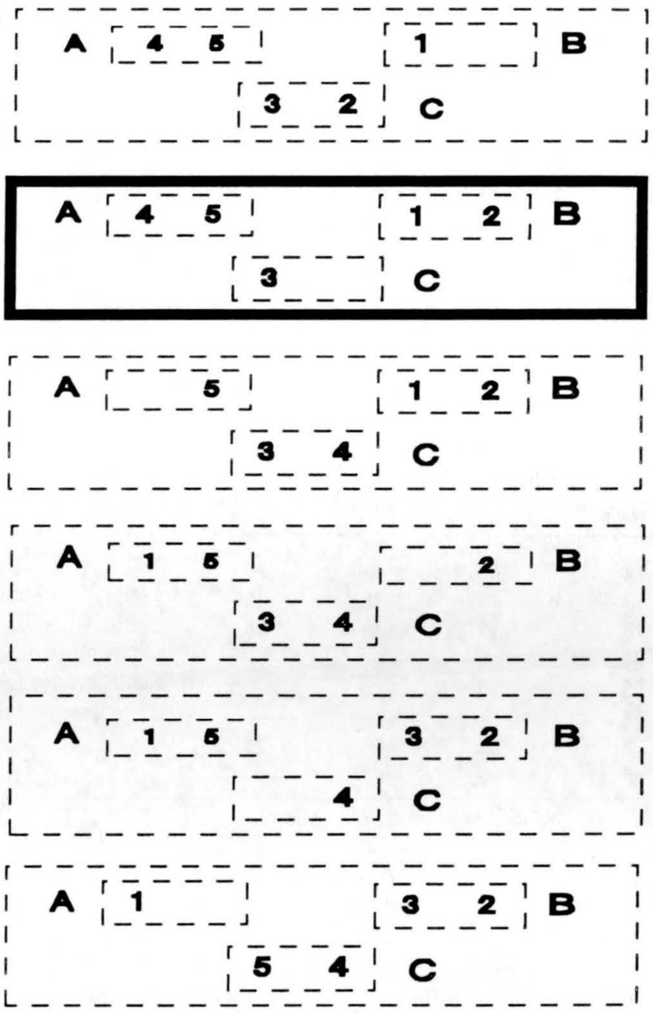
Phase I, Initialization, requires $O(\max(n \ln n, mn))$ time. The initialization time is dominated by the permutation time, and the amount of time required to make

Table 4 Major Iteration #3

(a) Action Set

	K				
Moves	1	2	3	4	5
1A	/	/	−10	—	—
1C	/	/	/	—	—
2A	0	—	—	—	—
2B	10	—	—	—	—
3A	/	/	−15	/	—
3B	/	/	/	0	—
4B	3	/	—	—	—
4C	/	−8	—	—	—
5B	−5	/	/	−5	/
5C	/	−10	/	/	−10
Swaps					
12	/	—	—	—	—
13	−18	−18	−18	—	—
14	−13	−13	—	—	—
15	−15	−15	−15	—	—
24	8	—	—	—	—
25	−10	—	—	—	—
34	−23	−23	—	—	—
35	−25	−25	−25	−25	—

(b)



(c) Algorithm Progress

Minor Iteration (k)	Best Move	Best Swap	Chosen Action	Save	p_k	p^*
1	2B	24	Move	10	10	10
2	4C	14	Move	−8	2	10
3	1A	15	Move	−10	−8	10
4	3B	35	Move	0	−8	10
5	5C	—	Move	−10	−18	10

assignments obeying capacity restrictions. Computation time for major iteration initialization is $O(n)$. As discussed earlier, the action set determination is the most critical.

Time complexity in the action set determination step is $O(n^2)$, dominated by the swap set creation. Identifying the best move and swap in each minor iteration requires $O(n^2)$ time. Sequence creation is negligible, independent of m and n. Step 3, solution refinement, requires $O(s^*)$ work, to finalize the set of moves and swaps accepted. The value s^* is certainly no more than n. Because the total number of minor iterations is bounded by n, the total work *per major iteration* is $O(n^3)$.

Empirical evidence indicates that the number of major iterations is weakly dependent on m and n, and is, in general, no more than ten. An upper bound on the number of major iterations, given a feasible initial solution, is $n(cmax - cmin)$, where $cmax$ is the maximum value of c_{ij}, and $cmin$ the minimum value.

To compare space utilization then, *VDSH*'s memory usage is 50% to 20% less than that of *MTG*, decreasing as the problem size increases. The additional space required by the optimization routine is a consequence of the branch-and-bound nature of the algorithm. The ratio of *HGAP* usage to that of *VDSH* is 30%. Because *VDSH* is a variable-depth technique, additional memory is required to maintain information carried from one minor iteration to the next.

4. Construction of a Testing System

To simplify and structure the generation and analysis of the experimental data points, a portable LCGAP testing system (LCGAPTS) is developed. LCGAPTS is organized into three components: (1) a random LCGAP generator; (2) a user-supplied suite of codes to solve randomly generated GAP instances (here we utilize *MTG*, *HGAP*, and *VDSH*); and (3) the data analysis module that collects the solution data and performs a statistical analysis to identify the relative efficiencies of the codes.

In summary, given a random seed number, a set of problem characteristics (levels of the experimental factors) (1) LCGAPTS creates a random GAP; (2) the GAP is solved by each of the GAP codes; and (3) solution data is collected in a convenient form for subsequent analysis.

5. The Experimental Design

In this section, we first review some of the major works in which rigorous statistical experimental design and analysis are applied to compare performance characteristics of heuristic and/or algorithmic alternatives. Next, we detail our experiment, design, and implementation.

5.1. Previous Work

The vital role of a sound statistical experimental design, along with some general guidelines for computational comparison of solution alternatives, has been discussed by Hoaglin and Andrews (1975), Crowder, Dembo, and Mulvey (1978), McGeoch (1986), Amini (1989), Greenberg (1990), Jackson et al (1990), Amini and Barr (1992). There are a number of sources with detailed discussions on the theoretical concepts, principles, and phases of experimental design, including the most recent books by Ostle and Malone (1988) and Mason, Gunst, and Hess (1989).

A survey by Jackson and Mulvey (1977) of published papers shows that a lack of understanding of careful experimental planning and reporting of experiments exists. A later study by Dembo and Mulvey (1978) suggests that a carefully-considered a priori experimental design is rarely applied, and offers a checklist of important factors in designing, implementing, and reporting a computational study. Later, revised guidelines to report empirical results are provided by Crowder, Dembo, and Mulvey (1978), Greenberg (1990), and Jackson et al (1990). Regardless of all these efforts, McGeoch (1986), Amini (1989), and Amini and Barr (1992) report that, while there is motivation for applying statistically experimental design in algorithmic performance analysis, it appears that not much progress has been made.

The most common method to analyze performance characteristics of algorithmic and/or heuristic alternatives has been the use of tables (graphs) of average measurement for each sample point, followed by an informal discussion of the results. There are a few instances in which application of statistical methods are formally considered. A discussion of these studies, incorporating different degrees of sophistication in their statistical analyses, is in order.

Comparing a new convex hull algorithm against two others, Eddy (1977) estimates standard deviations associated with the number of operations. Hart (1984) establishes 90% confidence intervals and conducts hypothesis testing in studying a binary search tree algorithm. Studying some hypothetical alternative heuristics on a sample of 15 traveling salesman problems, Golden and Stewart (1985) use two nonparametric tests, the Wilcoxon Signed Rank and Freidman tests. Also, they apply an *Expected Utility Approach* to identify the best heuristic as the one that performs well on average and that very rarely performs poorly. Analysis of variance (ANOVA) is applied by Moore and Whinston (1966) to measure the significance of quadratic programming algorithm parameters and problem attributes against computational time and iterations. The absence of a comprehensive a priori experimental design is a common factor among all the aforementioned works.

In a few instances an a priori statistical experimental design has appeared. Lin and Rardin (1980) present controlled statistical experimental design techniques for a comparison of integer programming algorithms. After a theoretical discussion on possible factorial designs to control seven nuisance problem parameters in the experiment, they offer a "blocking on problem" design along with ANOVA to compare two ILP algorithms on a set of 512 randomly generated problems. They conclude that there is room for much more research on the use of experimental design in mathematical programming. Hoaglin, Klema, and Peters (1982) investigate the performance of five nonlinear optimization routines in solving one test problem, starting from each of twenty randomly chosen starting points. Applying exploratory data analysis techniques and statistical models, the variability of performance across optimizers is described, and the effect of starting points is exposed. With the use of an a priori "split-plot" design and ANOVA, Amini and Barr (1992a, 1992b) conduct a rigorous computational study on the performance characteristics of the network reoptimization techniques under the impact of seven interacting experimental factors.

Although, the above studies apply common principles of statistical experimental designs, the related experimental designs are situation-specific and highly interrelated with the types of questions to be answered. In Golden and Stewart (1985), Lin and Rardin (1982), and Hoaglin, Klema, and Peters (1982) the quests are to identify an algorithm or heuristic alternative that under *all* circumstances has a better *average* performance than the others. What about the cases in which average performances of algorithmic / heuristic alternatives *significantly* changes from one factorial combination to another one? Or, what about the cases in which a researcher is interested in studying the heuristic / algorithmic behavior under a specific factorial combination, rather than the overall behavior? Amini and Barr (1992a and 1992b) address this issue by selecting a more flexible design, split-plot, to prepare a menu of network reoptimization algorithms for different factorial combinations. The same flexibility is required to provide answers for the questions raised in §1; hence, we apply a similar experimental design in this study.

5.2. The Experimental Environment

Following the principles suggested in the aforementioned studies, we present a carefully devised experimental design for computational comparison of the four prominent *LCGAP* optimization and heuristic methodologies with the *VDSH*. In designing the experiments for comparative analysis, the goal is to study the relative efficiency of the leading solution methods under the effects of the problem characteristics, singly and in combination. Hence, answers are provided to the following questions: (1) Is there a best overall method for solving *LCGAP*? (2) What are the effects of type and degree of parametric change on the performance of each solution methodology? (3) What are the effects of problem set and size on the performance of each method? (4) What are the interaction effects on the solution techniques when the above factors are changed singly or in combination?

The *factors* considered to be essential in the computational comparison are: problem class (capacities, load requirements, assignment costs); problem size (number of assignees, number of assignments); and solution methods. The *factor levels* are as follows: class (A, B, C, D); problem size (small and large); and five solution methods.

The experiment consists of four classes of test problems, accepted as the standard *LCGAP* library (Ross and Soland (1975), Fisher et al (1986), Martello and Toth (1987)). These classes are defined as follows:

Set A: *r_{ij} selected from a uniform distribution between 5 and 25; c_{ij} selected from a uniform distribution between 1 and 40;*

$$b_i = .6\left(\frac{n}{m}\right)15 + .4 \max_i \sum_{j \in J_i} r_{ij};$$

$$\textit{where } J_i = \{j: c_{ij} = \max_k (c_{kj})\}.$$

Set B: *same as A, except b_i is set to 70% of the value given in A.*

Set C: *same as A, except $b_i = .8 \sum_j r_{ij}/m$.*

Set D: *r_{ij} selected from a uniform distribution between 1 and 100; $c_{ij} = 111 - r_{ij} + e$, where e is uniform between -10 and 10;*

$$b_i = .8 \sum_j r_{ij}/m.$$

In developing problem set characteristics, two items are of concern. One is degree of solvability. A problem set that admits few feasible solutions is able to test the performance characteristics of a method more so than a set that admits many. Second, a problem set should attempt to characterize real-world conditions. In these regards, set A represents a class admitting many solutions, with a fairly simplistic assumption about cost/load relationships. Sets B and C provide tighter solution environments, again with little attention to reflecting the real world. Class D attempts to do both, by correlating cost and load size.

Within each class, problems are categorized as "small" (n = 10, 20, m = 3, 5) or "large" (n = 50, 100, 200, and m = 5, 10, 20). Since the *GAP* is NP-Complete, the performance of optimization methods degrades severely with increased problem size. As a result, these "small" test problems were applied in studies involving optimization techniques. To more adequately reflect real-world conditions, the "large" problems are included in this study. This is also of particular importance in comparing heuristic alternatives.

The solution Central Processing Unit (CPU) time is chosen to be the *dependent* (*response*) *variable*, for the following reasons. CPU time has long been a well-accepted standard of comparison in the mathematical programming community. In a practical setting, memory is commonly available, and solution quality and speed are of major concern. Moreover, the time to derive a solution is more sensitive to the factors discussed above than is the memory requirement.

Table 5 Split-plot Design

Problem	Problem Size		Heuristic	
Class	n	m	HGAP	VDSH
A	10	3		
		5		
	20	3		
		5		
B	10	3		
		5		
	20	3		
		5		
C	10	3		
		5		
	20	3		
		5		
D	10	3		
		5		
	20	3		
		5		

* 10 observations per cell.

The experiment's characteristics lend themselves to a *split-plot* design. Table 5 depicts the experimental design. This design is nested in the sense that within each treatment combination there are several treatment subcombinations. The underlying principle of the design is this: *whole* plots or *main* plots to which levels of one or more factors are applied, are divided into subplots to which one or more additional factors are applied. In this type of design, we are concerned with ranked-precision of information on factors, with the main plots having higher precision than the split plots.

Achieving the main goal of the study and answering the questions in §4 necessitate comparisons of the solution methodologies under the different treatment combinations as defined by the experimental design.

This includes a comprehensive analysis of the effects of the factors—singly and jointly—on the performance of each method. In particular, the objective is to identify the importance of the factors and their interactions, in terms of the magnitude of their effects on the solution CPU times generated by the codes.

We decide on ten *replications* (*observations per cell*) per split-plot design cell in order to generate an adequate number of degrees of freedom for the subsequent statistical analyses. Another related issue is the fashion in which factorial combinations (cells) are generated, ensuring randomness. As Gilsinn et al (1977) report, CPU time is affected by both time of day and job mix in a multi-tasking environment. In order to avoid biasing the results in this study, we randomly select combinations of the experimental factors (cells) and create the ten replications.

The statistical model used to relate the CPU times to the factors and sources of error encapsulated by the split-plot design is:

$$T_{ijkl} = \mu + P_i + S_j + U_k(ij) + M_l + P_i S_j + P_i M_l + S_j M_l + P_i S_j M_l + E_{ijkl}, \quad (5)$$

$\mu \equiv$ the mean CPU times,
$P_i \equiv$ the effect of problem set i, $i = 1, 2, 3, 4$,
$S_j \equiv$ the effect of problem size j, $j = 1, 2, 3, 4$,
$U_{k(ij)} \equiv$ the effect of problem k of set i and size j, $k = 1, 2, 3, 4$,
$M_l \equiv$ the effect of solution method l, $l = 1, 2$
$P_i S_j$, $P_i M_l$, $S_j M_l$, and $P_i S_j M_l \equiv$ the subplots effects, and
$E_{ijkl} \equiv$ the error term.

This model includes four subplot-factor interaction terms of two- and three-factor combinations, each of which may affect the response variable.

6. Design Implementation

The only implementations available to us were the optimization code, *MTG*, and the heuristic code, *HGAP*, developed by Martello and Toth (1987). This made it possible to collect data on the performance characteristics of both *MTG* and *HGAP* and conduct a rigorous statistical comparison with *VDSH*. To conduct the experimentation, two issues are addressed. (1) Lack of access to the computer codes of the optimization methods devised by Ross and Soland (1975), and Fisher et al (1986) made it impossible to generate the necessary data for an in-depth statistical comparison; (2) furthermore, it is known that creating an efficient GAP optimization method cannot exist unless $P = NP$. Thus, our experimentation with the three optimization methods under study and *VDSH* focuses on the "small" test problems along with an average performance comparison, while studying performance characteristics of the two heuristics incorporates a rigorous statistical evaluation of both the "small" and "large" test problems.

Hence, three experiments are conducted as follows. The first experiment is devoted to the performance comparison of the three leading optimization methods and *VDSH* on "small" test problems. In the second experiment, we provide a rigorous statistical comparison of *MTG* and *VDSH* on the same "small" set of test problems. In the third experiment, we compare *VDSH* and *HGAP* on the set of "large" test problems.

In the following experiments, all the test problem sets are generated and solved on the VAX 6420 computer under the VAX/VMS 5.4–2 operating system. The VAX Fortran compiler with optimization level 3 is utilized. A discussion of the three experiments along with the statistical methods, and analyses of the computational results are provided in the following sections.

6.1. First Experiment and Analysis

This experiment is concerned with the performance characteristics of the three GAP optimization methods and *V DSH* on the four classes of "small" test problems. First, we present the details of experiment and then our analysis.

The data points required for the three leading optimization methods and the *HGAP* in the first experiment are generated as follows: (1) we randomly ordered the problem classes and sets; (2) given each ordered set *LCGAP Testing System* is used to generate ten random replications; and (3) solve the replicates by *MTG* (Martello and Toth (1990)) and *VDSH* codes in a random order. This process is not applied with the Ross and

Soland (1975) and Fisher et al (1986) methods, due to the lack of the access to their computer codes. Hence, we converted the average solution times of the same set of problems solved by the two other methods from the CDC 6600 to DEC 10, using a conversion factor of five (Martello and Toth (1987)). Then applying a conversion factor of 1/7, provided by the Digital Equipment Company, the DEC 10 CPU times were converted to the VAX 6420. Although the validity of the conversion process when different hardware, operating systems, and computational environment involved is questionable, under the circumstances where computer codes for some of the solution methods under study are not available it becomes the only way for comparative study.

Hence, implementation of the split-plot design and detailed statistical analysis become infeasible; with two of the four methods, generation of the CPU times for individual problem instances—and the subsequent detection of significant variations and interactions between the experimental factors—is impossible. Consequently, a simple *average* CPU time comparison is applied.

The average CPU times and the relative error of the *VDSH* are shown in Table 6. Lower bounds are calculated by relaxing the capacity constraints; hence, errors are conservatively estimated. On problem classes A and C the solutions obtained by the *VDSH* are optimal. On class B problems, an average error of 7% is detected, while an average error of 240% is found for problems of class D, by far the most difficult problem set. For these "small" problems, *VDSH* exhibits very little sensitivity with respect to the problem class; the three optimization algorithms, however, did vary greatly in average solution time, and in fact there is no single algorithm that is dominant. In all classes, *VDSH* performs favorably compared to the leading algorithm. In the computationally intensive classes, B and D, *VDSH*'s solution times were much lower.

6.2. Second Experiment and Analysis

The second experiment's objective is to compare performance characteristics of one of the optimization methods, *MTG*, with *VDSH*. Since both computer codes are available, the aforementioned split-plot design is

Table 6 Performance Comparison—CPU-seconds and Relative Error

Problem	Problem Size		Ross-Soland	Fisher et al.	MTG	VDSH	
Class	*n*	*m*	Ave. Time	Ave. Time	Ave. Time	Ave. Time	Rel. Error
A	10	3	0.01	0.02	0.01	0.01	0.00
		5	0.02	0.04	0.03	0.01	0.00
	20	3	0.02	0.04	0.03	0.02	0.00
		5	0.06	0.09	0.01	0.03	0.00
B	10	3	0.16	0.22	0.11	0.01	0.00
		5	0.16	0.28	0.07	0.01	0.12
	20	3	23.47	0.73	149.32	0.05	0.06
		5	ETL	2.13	4.92	0.03	0.10
C	10	3	0.23	0.24	0.18	0.01	0.00
		5	0.26	0.36	0.07	0.01	0.00
	20	3	43.70	0.80	3.61	0.06	0.01
		5	ETL	1.96	5.90	0.04	0.00
D	10	3	0.57	0.26	0.29	0.00	1.81
		5	0.97	0.56	0.55	0.01	3.89
	20	3	ETL	2.77	11.72	0.02	1.47
		5	ETL	9.66	72.60	0.01	2.43

* VAX 6420 CPU seconds.

* ETL: Exceeded time limit imposed by Martello and Toth (1987).

applied, and a rigorous statistical analysis is conducted on all four classes of the "small" test problems.

Applying the *LCGAP Testing System*, the data generation for the second experiment includes: (1) randomly ordering the 32 experimental factor combinations (split-plot cells); (2) generating ten random "small" test problems for each ordered factorial combinations; and (3) solving in random order the ten random test problems by each of the two codes and recording the solution CPU times. Hence, a total of 160 random test problems are generated and solved by both codes.

The statistical method required is *analysis of variance* (*ANOVA*). This method provides information for testing simultaneously the significance of the difference between mean solution times of the methods under single or multiple factor treatment combinations. Given the experimental design, the significance of the difference between treatment-combination mean times could be tested by analyzing the variance within and between the samples.

Table 7 ANOVA Table for CPU-Times—Small Problems

(a) includes test problems not found feasible

Source	DF	SS	MS	F	P-Value
P	3	21501.139	7167.046	3.86	0.0109
S	3	20999.961	6999.987	3.77	0.0122
$P \times S$	9	60961.497	6773.500	3.64	0.0004
$U \times (P \times S)$	144	267658.418	1858.739	1.00	0.5002
M	1	8406.230	8406.230	4.52	0.0352
$P \times M$	3	21485.526	7161.842	3.85	0.0109
$S \times M$	3	20947.889	6982.630	3.76	0.0124
$P \times S \times M$	9	60932.121	6770.236	3.64	0.004

(b) excludes test problems not found feasible

Source	DF	SS	MS	F	P-Value
P	3	334.666	111.555	2.60	0.0546
S	3	278.998	92.999	2.17	0.0945
$P \times S$	9	577.964	64.218	1.50	0.1547
$U \times (P \times S)$	141	6254.917	44.361	1.04	0.4207
M	1	186.063	186.063	4.34	0.0391
$P \times M$	3	302.196	100.732	2.35	0.0753
$S \times M$	3	202.863	67.621	1.58	0.1977
$P \times S \times M$	9	750.156	83.351	1.95	0.0508

Source: Source of problem variation—single and interaction terms.
DF: Degrees of freedom.
SS: Sum of squares.
MS: Mean squared.
F: F-value.
P-value: P-value.

The analysis of variance is initiated by a translation of the objectives of the study into statistical hypotheses. The hypotheses were categorized into two main groups: hypotheses to detect significant difference between single factor means, and hypotheses to study the significant differences between the multiple-factor interaction means. For example, the null hypotheses with regard to the problem size factor can be stated as: $\mu_{P(i)} = 0$, for all i. The associated alternative hypothesis is: $\mu_{P(i)} \neq 0$, for at least two is. Hypotheses can be stated similarly for each of the other single and multiple-factor interactions. The significance level selected prior to the analysis was 5%.

Two ANOVA are conducted. In the first analysis, the solution times associated with the test problems not found feasible are included. Table 7(a) summarizes this information. The null hypotheses associated with the problem class, size, heuristics, and interactions of these three factors were rejected even at much smaller significance levels. Thus, at least two of the mean solution times stated in each null hypotheses are significantly different. In terms of relative performance, this does not permit ranking of the heuristics under different factor combinations.

When comparing more than two means, an ANOVA procedure indicates whether the means are significantly different from each other, but it does not show which means actually differ. The significance shown by our ANOVA makes it desirable to conduct further analyses to determine which pairs or groups of solution average CPU times are significantly different. Such comparisons between means are sometimes referred to as *mean comparisons*. Rejection of the null hypotheses necessitates mean comparisons to provide detailed information about the observed differences in means.

Tukey's Significance Test is applied to compare and rank the performance of the heuristics under the effect of different single-factor levels as well as various treatment combinations. Tukey's test controls the *experi-*

Table 8(a) **Ave. Solution CPU Time For MTG and VDSH—Small Problems**

	Problem Size		Solution Method	
Problem Class	*n*	*m*	MTG	VDSH
A	10	3	0.01	0.01
		5	0.03	0.01
	20	3	0.08	0.02
		5	0.01	0.03
B	10	3	0.12	0.01
		5	0.08	0.01
	20	3	149.33	0.05
		5	4.92	0.03
C	10	3	0.19	0.01
		5	0.07	0.01
	20	3	3.61	0.06
		5	5.90	0.04
D	10	3	0.29	0.00
		5	0.55	0.01
	20	3	11.72	0.02
		5	72.60	0.01

* VAX 6420 CPU seconds.
* Includes test problems not found feasible.

mentwise error rate (*EER*) for multiple comparisons, defined as the probability of rejecting one or more of the null hypotheses when making statistical tests of two or more null hypotheses. In this study, having multiple mean comparisons requires more control on EER, and consequently the use of Tukey's test.

Applying Tukey's test, we conclude the following major result on the "small" test problems: regardless of problem class, size, or both, *VDSH*, with an overall average solution time of 0.022 CPU seconds, significantly outperforms *MTG*, with an overall average solution time of 10.273 CPU seconds. A comparison of average solution times based on the problem class or problem size indicates that *V DSH* dominates *MTG*. Table 8(a) shows the detailed average solution times.

In the second ANOVA, solution times on the infeasible test problems are excluded and treated as "missing" data points. Table 7(b) depicts the information provided by the second ANOVA procedure. It is found that the null hypotheses associated with the problem class, problem size, algorithm, and the three-factor interaction are rejected. Again to identify the significant differences among the solution times, Tukey's test is applied and the following result is obtained on the "small" test problems: the *V DSH* (overall average solution time of .0199 CPU seconds) significantly outperforms *MTG* (overall average solution time of 1.8499 CPU seconds) in solving all problem classes and sizes. Table 8(b) presents the detailed average solution times.

6.3. Third Experiment and Analysis

To compare the relative performances of the *HGAP* and the *VDSH* on the "large" size problems, the third experimental design is developed. In this design, the same factors and levels included in the second design are considered, except that the problem size levels increases from four to nine (as in Martello and Toth (1987)). As a result, the number of factorial combinations increased from 32 to 72 and the number of test problems generated and solved is increased from 320 to 720. The nine

Table 8 (b) **Ave. Solution CPU Time for MTG and VDSH—Small Problems**

			Solution Method			
	Problem Size		MTG		VDSH	
Problem Class	*n*	*m*	# Exclusions	Ave. Time	# Exclusions	Ave. Time
A	10	3	0	0.01	0	0.01
		5	0	0.03	0	0.01
	20	3	0	0.08	0	0.02
		5	0	0.01	0	0.03
B	10	3	3	0.06	2	0.01
		5	0	0.08	0	0.01
	20	3	3	18.72	1	0.05
		5	0	4.92	0	0.03
C	10	3	3	0.07	1	0.01
		5	0	0.07	0	0.01
	20	3	0	3.61	2	0.04
		5	0	5.90	0	0.04
D	10	3	0	0.01	0	0.01
		5	0	0.01	0	0.01
	20	3	0	0.00	0	0.02
		5	0	0.01	0	0.03

* VAX 6420 CPU seconds.
* Excludes test problems not found feasible.

Table 9 (a) Ave. Solution CPU Time for HGAP and VDSH—Large Problems

Problem Class	Problem Size		Solution Method	
	n	m	HGAP	VDSH
A	50	5	0.02	0.19
		10	0.03	0.18
		20	0.06	0.25
	100	5	0.05	1.07
		10	0.06	1.12
		20	0.09	1.57
	200	5	0.14	9.08
		10	0.19	11.00
		20	0.25	11.78
B	50	5	0.02	0.82
		10	0.04	0.52
		20	0.06	0.34
	100	5	0.06	11.59
		10	0.08	5.25
		20	0.13	3.84
	200	5	0.17	131.91
		10	0.23	54.89
		20	0.36	45.69
C	50	5	0.02	0.90
		10	0.04	1.10
		20	0.08	1.10
	100	5	0.06	12.13
		10	0.08	5.59
		20	0.16	9.20
	200	5	0.17	131.54
		10	0.24	47.76
		20	0.40	79.42
D	50	5	0.01	0.12
		10	0.01	0.52
		20	0.01	0.24
	100	5	0.01	0.98
		10	0.01	1.51
		20	0.01	1.64
	200	5	0.03	9.71
		10	0.03	11.03
		20	0.05	13.27

* VAX 6420 CPU seconds.

* Includes test problems not found feasible.

configurations evaluated are generated with three values of $n(50, 100, 200)$, and three values of $m(5, 10, 20)$. *LCGAP Testing System* applies the same random process as described in the second experiment to generate and solve "large" test problems within the four problem classes.

The statistical model associated with the third experimental design is the same as the second one, except that the problem set is "large," the number of problem sizes increased to nine, and *HGAP* is substituted for *MTG*. As in case one: (1) the same set of hypotheses regarding the relative performances of the *HGAP* and *VDSH* are established; and (2) tested by two ANOVA procedures. Tables 9(a) and 9(b) show the average solution times of *HGAP* and *VDSH*. Also, a summary of the two ANOVA results are depicted in Table 10 at 5% significance level.

Table 10(a) presents the results of ANOVA of the CPU times including the solution times associated with the problems not found feasible. Except in one case, all the null hypotheses are rejected at even a lower significance rate (*p-value* = .0001). As a result, Tukey's test is applied and we find that *HGAP*, with an average solution time of .0963, significantly outperforms the *VDSH*, with an average solution time of 17.1896 CPU seconds. Excluding the solution times of the problems that are not found feasible from the data set and applying ANOVA resulted in the information that is depicted in Table 10(b). As in the previous case, all the null hypotheses are rejected. And again, regardless of the problem class or size, Tukey's test indicates that *HGAP* (overall average solution time = 0.0990) dominates *VDSH* (overall average solution time = 10.6338) on the "large" test problem sets. However, the significance of the difference is weaker than in the previous scenario.

Although the experimentation with the "large" test problems strongly suggests the superior performance of the *HGAP*, it should be noted that the solution quality provided by this method is inferior to the *VDSH*. The *VDSH* paid a price of 10 to 17 CPU seconds to achieve an average of 20% improvement in solution quality over *HGAP*, where relative errors are calculated with respect to the lower bound. Hence, from this point of view the superior performance of the *VDSH* can be verified. Table 11 shows the improvement of solution quality for each class and size of the "large" test problems.

It should be noted that "statistical significance" does not imply practical importance in all cases (see Snedecor

Table 9 (b) Ave. Solution CPU Time for HGAP and VDSH—Large Problems

			Solution Method			
	Problem Size		HGAP		VDSH	
Problem Class	*n*	*m*	# Exclusions	Ave. Time	# Exclusions	Ave. Time
A	50	5	0	0.02	0	0.19
		10	0	0.03	0	0.18
		20	0	0.06	0	0.25
	100	5	0	0.05	0	1.07
		10	0	0.06	0	1.12
		20	0	0.09	0	1.57
	200	5	0	0.14	0	9.08
		10	0	0.19	0	11.00
		20	0	0.25	0	11.78
B	50	5	0	0.02	1	0.69
		10	0	0.04	0	0.52
		20	0	0.06	0	0.34
	100	5	0	0.06	2	11.00
		10	0	0.08	0	5.25
		20	0	0.13	0	3.84
	200	5	0	0.17	9	63.72
		10	0	0.23	0	54.89
		20	0	0.36	0	45.69
C	50	5	0	0.02	2	0.69
		10	0	0.04	0	1.10
		20	0	0.08	0	1.10
	100	5	0	0.06	5	11.51
		10	0	0.08	0	5.59
		20	0	0.16	0	9.20
	200	5	0	0.17	10	—
		10	0	0.24	0	47.76
		20	0	0.40	0	79.42
D	50	5	0	0.01	0	0.12
		10	0	0.01	0	0.52
		20	0	0.01	0	0.24
	100	5	0	0.01	0	0.98
		10	0	0.01	0	1.51
		20	0	0.01	0	1.64
	200	5	0	0.03	0	9.71
		10	0	0.03	0	11.03
		20	0	0.05	0	13.27

* VAX 6420 CPU seconds.

* Excludes test problems not found feasible.

and Cochran (1976)). First, in terms of computational resource availability, paying 10 to 17 CPU seconds to gain 20% improvement in the solution quality of an *NP-Hard* problem may be considered "insignificant." Secondly, there is the issue of data integrity. In some instances, errors may arise in the estimation of problem

Table 10 **ANOVA Table for CPU-Times—Large Problems**

(a) includes test problems not found feasible

Source	DF	SS	MS	*F*	*P*-Value
P	3	30960.619	10320.206	235.17	0.0001
S	8	95828.830	11978.604	272.97	0.0001
$P \times S$	24	68632.705	2859.696	65.17	0.0001
$U \times (P \times S)$	4	14210.678	43.860	1.00	0.5019
M	1	52592.597	52592.597	1198.47	0.0001
$P \times M$	3	30605.974	10201.991	323.48	0.0001
$S \times M$	8	94990.047	11873.756	270.58	0.0001
$P \times S \times M$	24	68422.739	2850.947	64.97	0.0001

(b) excludes test problems not found feasible

Source	DF	SS	MS	*F*	*P*-Value
P	3	7114.828	2371.609	87.73	0.0001
S	8	31698.678	3962.335	146.57	0.0001
$P \times S$	24	17754.544	739.773	27.36	0.0001
$U \times (P \times S)$	324	9781.949	30.191	1.12	0.1665
M	1	18376.070	18376.070	679.75	0.0001
$P \times M$	3	8502.468	2834.156	104.84	0.0001
$S \times M$	8	31126.885	3890.861	143.93	0.0001
$P \times S \times M$	23	16923.476	735.803	27.22	0.0001

Source: Source of problem variation—single and interaction terms.
DF: Degrees of freedom.
SS: Sum of squares.
MS: Mean squared.
F: *F*-value.
P-value: *P*-value.

parameters. In such a case, the importance of being close to the mathematical optimum is diminished.[1]

7. Summary and Conclusions

In this paper first we present a new heuristic, Variable-Depth-Search (*VDSH*), to solve the Linear Cost Generalized Assignment Problem (LCGAP). Next, we review application of statistical experimental design in comparing performance characteristics of algorithmic and heuristic alternatives. Applying a rigorous statistical design of experiment, split-plot design, we compare VDSH against the four leading GAP heuristics and optimization algorithms. In-depth statistical analyses (analysis of variance and Tukey's test) are followed to

Table 11 **Solution Quality—Large Problems Relative Error ((result-lower.bound)/(lower.bound))**

	Problem Size		Solution Method			
			HGAP		VDSH	
Problem Class	*n*	*m*	# Exclusions	Error	# Exclusions	Error
A	50	5	0	0.021	0	0.014
		10	0	0.027	0	0.026
		20	0	0.045	0	0.043
	100	5	0	0.004	0	0.004
		10	0	0.007	0	0.007
		20	0	0.006	0	0.006
	200	5	0	0.003	0	0.002
		10	0	0.003	0	0.002
		20	0	0.002	0	0.002
B	50	5	0	0.463	1	0.334
		10	0	0.399	0	0.258
		20	0	0.354	0	0.275
	100	5	0	0.329	2	0.201
		10	0	0.434	0	0.205
		20	0	0.205	0	0.111
	200	5	0	0.388	9	0.112
		10	0	0.498	0	0.216
		20	0	0.273	0	0.100
C	50	5	0	0.473	2	0.349
		10	0	0.708	0	0.448
		20	0	1.341	0	0.924
	100	5	0	0.312	5	0.186
		10	0	0.647	0	0.362
		20	0	1.038	0	0.506
	200	5	0	0.347	10	—
		10	0	0.546	0	0.294
		20	0	0.705	0	0.280
D	50	5	0	1.568	1	1.511
		10	0	3.747	1	2.598
		20	0	3.758	5	3.615
	100	5	0	1.591	0	1.518
		10	0	2.813	0	2.661
		20	0	3.834	0	3.834
	200	5	0	1.539	0	1.406
		10	0	2.691	0	2.517
		20	0	4.040	0	3.815

* Excludes test problems not found feasible.

[1] As noted by one of the anonymous referees.

evaluate the relative efficiencies of VDSH and the leading GAP solution methods under different factorial combinations.

We show that on "small" test problems, *VDSH* produces solutions of comparable quality and its average solution time is "significantly" smaller than the leading algorithms. On the set of "large" test problems, although *VDSH* performance is inferior to the leading heuristic, *HGAP*, however, it improves the solution quality by 20% over *HGAP*, given the correct problem parameters. A major trade-off between *VDSH* and *HGAP* becomes the solution quality versus solution time.[2]

[2] We thank the anonymous area editor and three referees for their careful review of the original manuscript and comments and suggestions that enhance the presentation and content of this paper.

References

Amini, M. M., "Network Reoptimization: A Computational Comparison of Algorithmic Alternatives," University Microfilms, International, Ann Arbor, MI (dissertation, Department of Operations Research, Southern Methodist University, Dallas, TX, December 1989).

—— and R. S. Barr, "Network Reoptimization Algorithms: A Statistical Designed Comparison," *ORSA J. Computing*, 5, 4 (1993), 395–408.

—— and R. S. Barr, "Network Reoptimization: A Computational Study of the Relaxation Method," Working Paper DS5-10-92, MIS/DS Department, The Fogelman College of Business and Economics, Memphis State University, Memphis, TN 38152, 1992b.

Balachandran, V., "An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks," Working Paper 34-72-3, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, November 1972.

Casco, D. O., B. L. Golden, and E. A. Wasil, "Vehicle Routing with Backhauls: Models, Algorithms, and Case Studies," *Vehicle Routing: Methods and Studies*, Elsevier Science Publishers, 1988.

Clarke, G. and J. W. Wright, "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Oper. Res.*, 12 (1964), 568–581.

Crowder, H. P., R. S. Dembo, and J. M. Mulvey, "Reporting Computational Experiments in Mathematical Programming," *Math. Programming*, 15 (1978), 316–329.

Dembo, R. S. and J. M. Mulvey, "On the Analysis and Comparison of Mathematical Programming Techniques," in W. W. White, (Ed.), *Computers and Mathematical Programming*, National Bureau of Standards Special Publication 502, 1978.

Eddy, W. F., "A New Convex Hull Algorithm for Planar Sets," *ACM Transactions on Mathematical Software*, 3, 4 (1977), 398–403.

Fisher, M. L. and R. Jaikumar, "A Generalized Assignment Heuristic for Vehicle Routing," *Networks*, 11 (1981), 109–124.

——, —— and L. N. Van Wassenhove, "A Multiplier Adjustment Method for the Generalized Assignment Problem," *Management Sci.*, 32, 9 (1986), 1095–1103.

Gavish, B. and H. Pirkul, "Algorithms for the Multi-Resource Generalized Assignment Problem," *Management Sci.*, 37, 6 (1991), 695–713.

Gilsinn, J., K. Hoffman, R. H. F. Jackson, E. Leyendecker, P. Saunders, and D. Shier, "Methodology and Analysis for Comparing Discrete Linear L_1 Approximation Codes," *Communications Statistics, Simulation, and Computations*, B6.4 (1977), 399–413.

Golden, B. L. and W. R. Stewart, "Empirical Analysis of Heuristics," in E. L. Lawler et al. (Eds.), *The Traveling Salesman Problem*, John Wiley and Sons, New York, 1985.

Greenberg, H. J., "Computational Testing: Why, How and How Much," *ORSA J. Computing*, 2 (1990), 94–97.

Grigoriadis, M. D., D. T. Tang and L. S. Woo, "Considerations in the Optimal Synthesis of Some Communication Networks," Presented at the Joint National Meeting of ORSA/TIMS, Boston, MA, April 1974.

Hall, R. W., "Route Choice on Networks with Concave Costs and Exclusive Arcs," *Transportation Res.*, 23b (1989), 103–121.

Hart, R. R., "The Average Height of Binary Search Trees," Master's Thesis, Department of Computer Sciences, University of California at Irvine, 1984.

Hoaglin, D. C. and D. F. Andrews, "The Reporting of Computation-Based Results in Statistics," *The American Statistician*, 29, 3 (1975), 122–126.

——, V. C. Klema and S. C. Peters, "Exploratory Data Analysis in a Study of the Performance of Nonlinear Optimization Routines," *ACM Transactions on Mathematical Software*, 8, 2 (1982), 145–162.

Jackson, R. H. F., P. T. Boggs, S. G. Nash, and S. Powell, "Report of Ad Hoc Committee to Revise the Guidelines for Reporting Computational Experiments in Mathematical Programming," *ORSA/CSTS Newsletter*, 10, 1 (1989), 7–14.

—— and J. M. Mulvey, "A Critical Review of Methods for Comparing Mathematical Programming Algorithms and Software (1953–1977)," Technical Paper, Department of Civil Engineering, Princeton University, Princeton, NJ, 1977.

Kernighan, B. and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *BSTJ*, 49 (1970), 291–307.

Lin, S., "Computer Solutions of the Traveling Salesman Problem," *BSTJ*, 44 (1965), 2245–2269.

—— and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Oper. Res.*, 21 (1973), 498–516.

Lin, B. W. and R. L. Rardin, "Controlled Experimental Design for Statistical Comparison of Integer Programming Algorithms," *Management Sci.*, 25, 12 (1980), 1258–1271.

Macgeoch, C. C., "Experimental Analysis of Algorithms," University Microfilms, International, Ann Arbor, MI (dissertation Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, August 1986).

Martello, S. and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley and Sons, 1990.

—— and ——, "Linear Assignment Problems," *Annals of Discrete Math.*, 31 (1987), 259–282.

Mason, R. L., R. F. Gunst, and J. L. Hess, *Statistical Analysis of Experiments*, John Wiley and Sons, 1989.

Moore, J. H. and A. B. Whinston, "Experimental Methods in Quadratic Programming," *Management Sci.*, 13, 1 (Series A) (1966), 58–76.

Ostel, B. and L. C. Malone, *Statistics in Research*, The Iowa State University Press, Ames, IA, 1988.

Papadimitriou, C. H. and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.

Racer, M., "Coordinating Inbound and Outbound Vehicle Routes Within a Decentralized Decision Environment," University Microfilms, International, Ann Arbor, MI (dissertation, Industrial Engineering and Operations Research Department, University of California at Berkeley, December 1990).

Ross, G. T. and R. M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Math. Programming*, 8 (1975), 92–103.

Snedecor, G. W. and W. G. Cochran, *Statistical Methods*, The Iowa State University Press, Ames, IA, 1976.

Accepted by Thomas M. Liebling; received February 3, 1992. This paper has been with the authors 5 months for 1 revision.